Numérique et science informatique
Classe de Terminale

Lycée Hoche

année scolaire 2024-2025

Contents

1 Introduction

L'algorithme de recherche dichotomique, présenté en première, permet la recherche d'un élément dans un tableau trié en temps logarithmique.
Il consiste à séparer le tableau en deux à chaque étape, et à chercher l'élément dans l'un des sous-tableaux.

L'approche « diviser pour régner » est une généralisation de la dichotomie. Un algorithme de type « diviser pour régner » se décompose en trois phases.

1. Diviser : étant donné le problème à résoudre, on découpe l'entrée en deux ou plusieurs morceaux.
2. Résoudre : à l'aide d'appels récursifs, on résout le problème sur chacun des morceaux.
3. Combiner : à partir des résultats des appels récursifs, on construit la solution du problème de départ.

Il est facile d'oublier, dans cette description, un aspect absolument fondamental : puisque l'on utilise des algorithmes récursifs, il ne faut pas oublier de traiter les cas de base.

2 Le tri fusion

En première ont été vus les tris par insertion et sélection, qui permettent de trier un tableau de nombres en temps quadratique : $O(n^2)$.

La méthode " diviser pour régner " permet de décrire un algorithme de tri de meilleure complexité, à savoir $O(nlog(n))$ pour un tableau de taille n. Le principe de l'algorithme suit les phases de la méthode " diviser pour régner ".

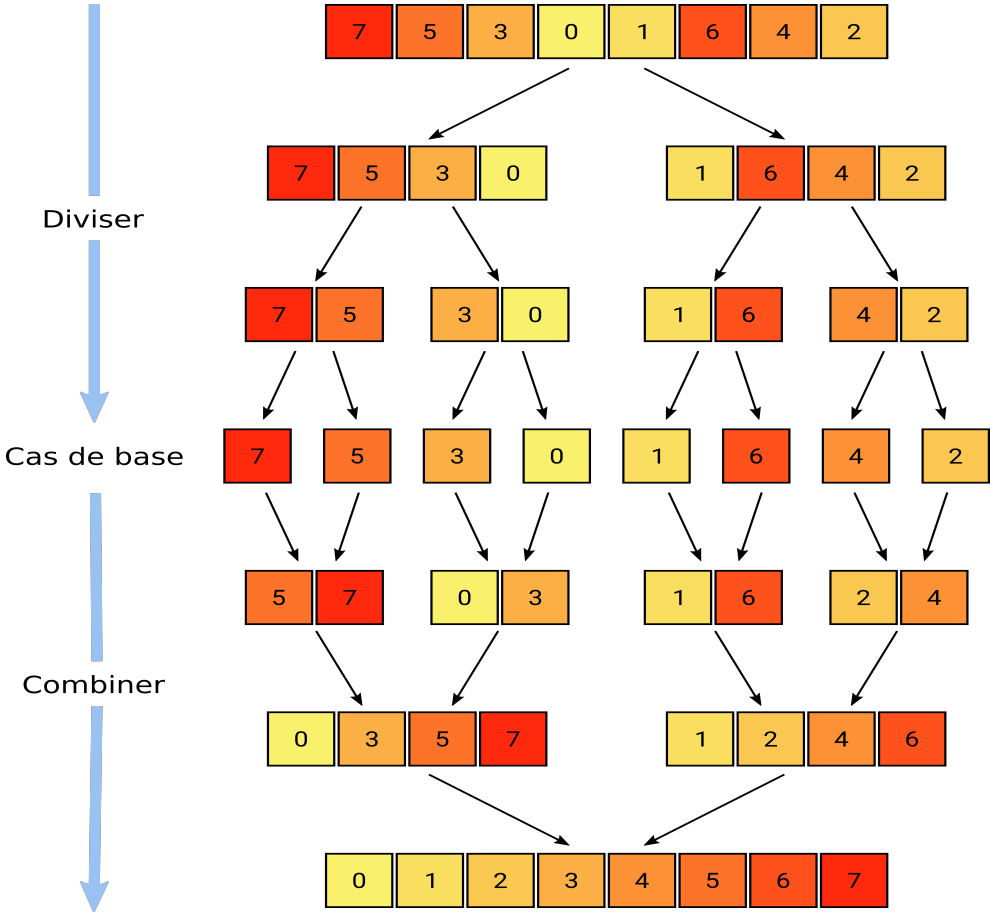
- Diviser : découpe du tableau initial en deux sous-tableaux de taille (environ) égale.
- Résoudre : tri des deux sous-tableaux, grâce à deux appels récursifs.
- Combiner : fusion des deux sous-tableaux triés pour produire le tableau trié complet.

La figure suivante illustre le tri fusion sur un tableau de taille 8.

L'étape délicate est la fusion des tableaux triés.

Pour la découpe , on coupe simplement le tableau par son milieu.S'il possède n éléments, les deux sous tableaux sont :

T1 = T[0 : n//2] et T2 = T[n//2 : n]



2.1 Fusion de tableaux triés

Le problème, qui constitue la brique de base du tri fusion, est le suivant : étant donnés deux tableaux T_1 et T_2 triés, il s'agit de construire le tableau T, lui aussi trié, qui contient l'ensemble des éléments présents dans T_1 et T_2 .

Cette brique de base n'utilise pas elle-même la méthode " diviser pour régner ".
On suppose que l'élément en haut de pile de T_1 et T_2 est leur plus petit élément.

Pendant l'algorithme, on ne peut donc accéder qu'à deux éléments : le plus petit de chaque pile.
L'algorithme consiste alors à itérativement dépiler un élément soit de T_1 soit de T_2 (selon lequel est le plus petit), pour l'empiler sur la pile de sortie T.

Si une des deux piles est vide, on dépile entièrement l'autre pile.

L'implantation de l'algorithme peut se faire directement avec des tableaux, sans implanter une véritable structure de pile. On utilise pour chacun des deux tableaux un indice qui indique le haut de pile : dépiler consiste simplement à incrémenter l'indice, et empiler à mettre à jour une valeur de T et incrémenter l'indice.
L'implantation la plus simple consiste à faire une première boucle qui s'occupe du cas où les deux piles sont encore non vides. On traite la fin (une des deux piles est vide) dans un second temps.

Complexité

Cet algorithme effectue un nombre de comparaisons linéaire en la taille du tableau fusionné .
En effet, à chaque étape de la première boucle, une comparaison est effectuée et l'indice est incrémenté de 1.

Correction

On utilise l'invariant de boucle suivant : à chaque entrée dans la première boucle while, les j premiers éléments de T sont triés et $T[j-1]$ est inférieur ou égal à $T_1[i_1]$ et $T_2[i_2]$.

Cet invariant est vérifié avant la première entrée puisque $T[j-1]$n'existe pas.
Si l'invariant est correct à l'entrée d'une itération, puisque $T[j-1]$ est inférieur ou égal à $T_1[i_1]$ et $T_2[i_2]$, les premiers éléments de T sont toujours triés après l'itération. Supposons que $T_1[i_1] < T_2[i_2]$,alors $T_1[i_1]$ est le $(j+1)$ème élément de T: il est bien inférieur $T_1[i_1+1]$ et $T_2[i_2]$, donc l'invariant reste valide après l'itération.

L'autre cas est symétrique. Avant de rentrer dans l'une des deux dernières boucles, les premiers éléments de T sont triés et inférieurs aux éléments non encore insérés de la pile non vide. La dernière boucle exécutée conserve bien le fait que T est trié. Ainsi, l'algorithme est correct.

2.2 L'algorithme du tri fusion

Étant donné l'algorithme de fusion, l'algorithme du tri fusion ne fait plus de difficulté. Il faut simplement ne pas oublier les cas de base.

```
def tri_fusion(T):
    '''
    Renvoie le tableau T trié
    '''

    n = len(T)
    # Cas de base
    if n <2:
        return T[:]  # copie de T

    # Diviser : découpe de T

    T1 = T[0 : n//2]
    T2 = T[n//2 : n]
    # Résoudre : appels récursifs
    T1 = tri_fusion(T1)
    T2 = tri_fusion(T2)
    # Combiner : fusion des tableaux triés
    return fusion(T1, T2)
```

Correction
La preuve de correction de l'algorithme est une preuve par récurrence sur la taille de T. Les cas de base (taille nulle ou 1) sont bien traités. Soit $n > 1$ et supposons par hypothèse de récurrence que l'algorithme trie correctement les tableaux de taille n.

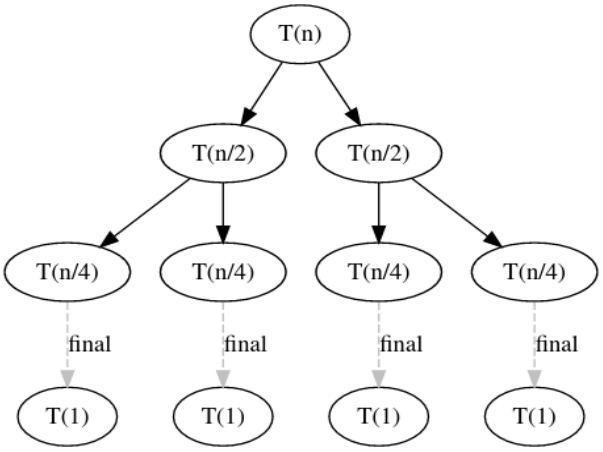
Puisque T_1 est de taille $\lfloor \frac{n}{2} \rfloor$ et T_2 de taille $n - \lfloor \frac{n}{2} \rfloor = \lfloor \frac{n}{2} \rfloor$, les deux tableaux T_1 et T_2 sont correctement triés par les appels récursifs. Puisque l'algorithme fusion est correct, le tableau renvoyé est bien trié.

Complexité:

Pour la complexité de l'algorithme, on se réfère à la figure d'illustration. On remarque d'abord que les divisions de tableaux en sous-tableaux sont « gratuites », au moins en termes de comparaisons. Il s'agit donc de borner le nombre de comparaisons effectuées lors des fusions.

On a dit dans la partie précédente que la fusion de deux tableaux dont la somme des tailles vaut m nécessite $O(m)$ comparaisons.

À chaque étage de fusion, la somme des tailles de tous les tableaux fusionnés est exactement n : le nombre de comparaisons effectuées à chaque étage est donc $O(n)$.
Il ne reste plus qu'à compter le nombre d'étages.



On arrive au dernier niveau de l'arbre en b étapes , ce qui correspond comme nous le verrons plus tard , à la hauteur de l'arbre.
L'égalité $n = 2^b$ (ce que l'on suppose au départ) implique $b = log_2(n)$.

Au total , on obtient une complexité de la "division" en $O(log_2(n))$.

Notons que si la taille du tableau est $l \neq 2^n$, on prend le plus petit entier n vérifiant $l \leq 2^n$.
Pour le tri_fusion , finalement , la complexité est alors au maximum en $O(log_2(n))$.

L'algorithme a donc une complexité $O(nlog_2(n))$.